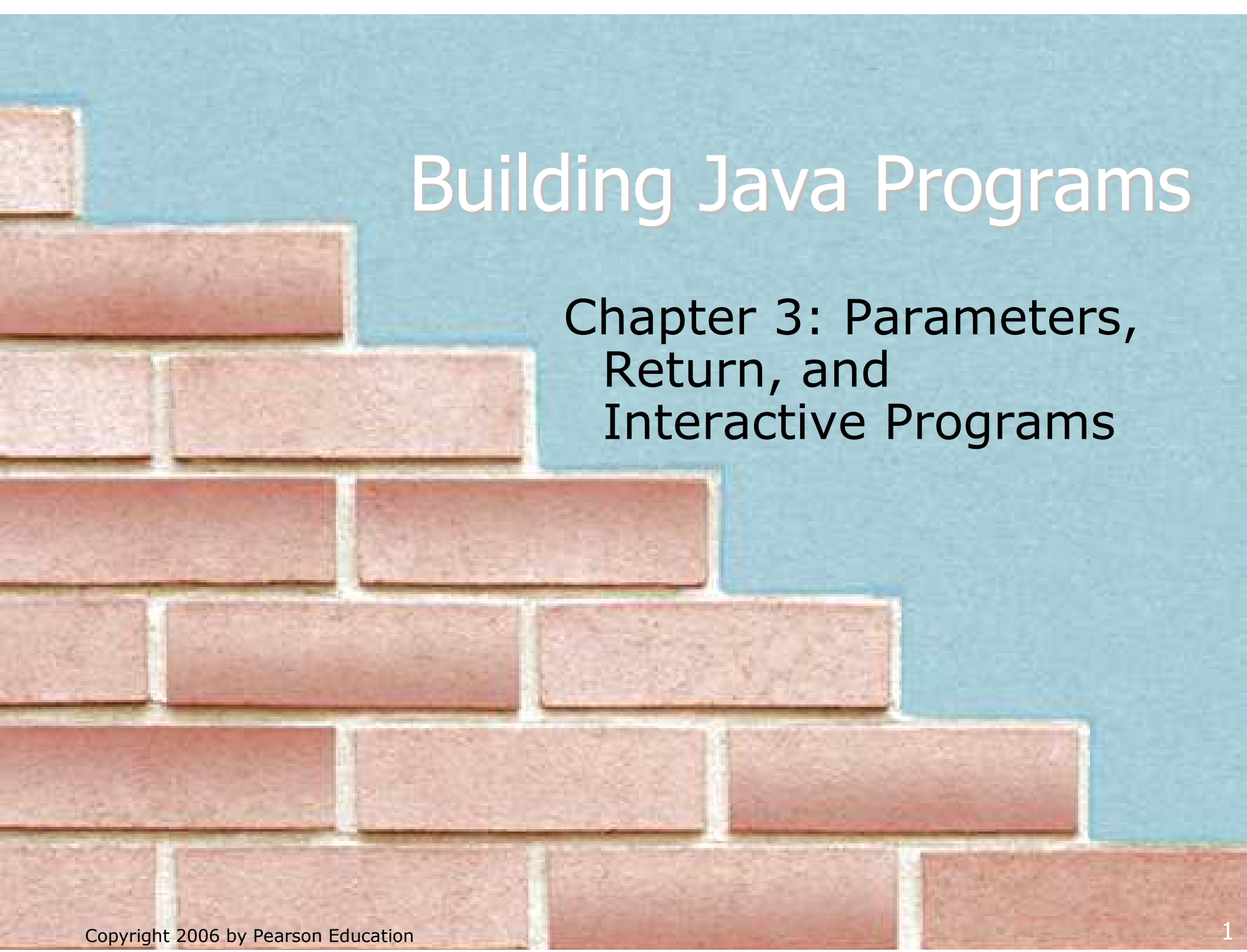
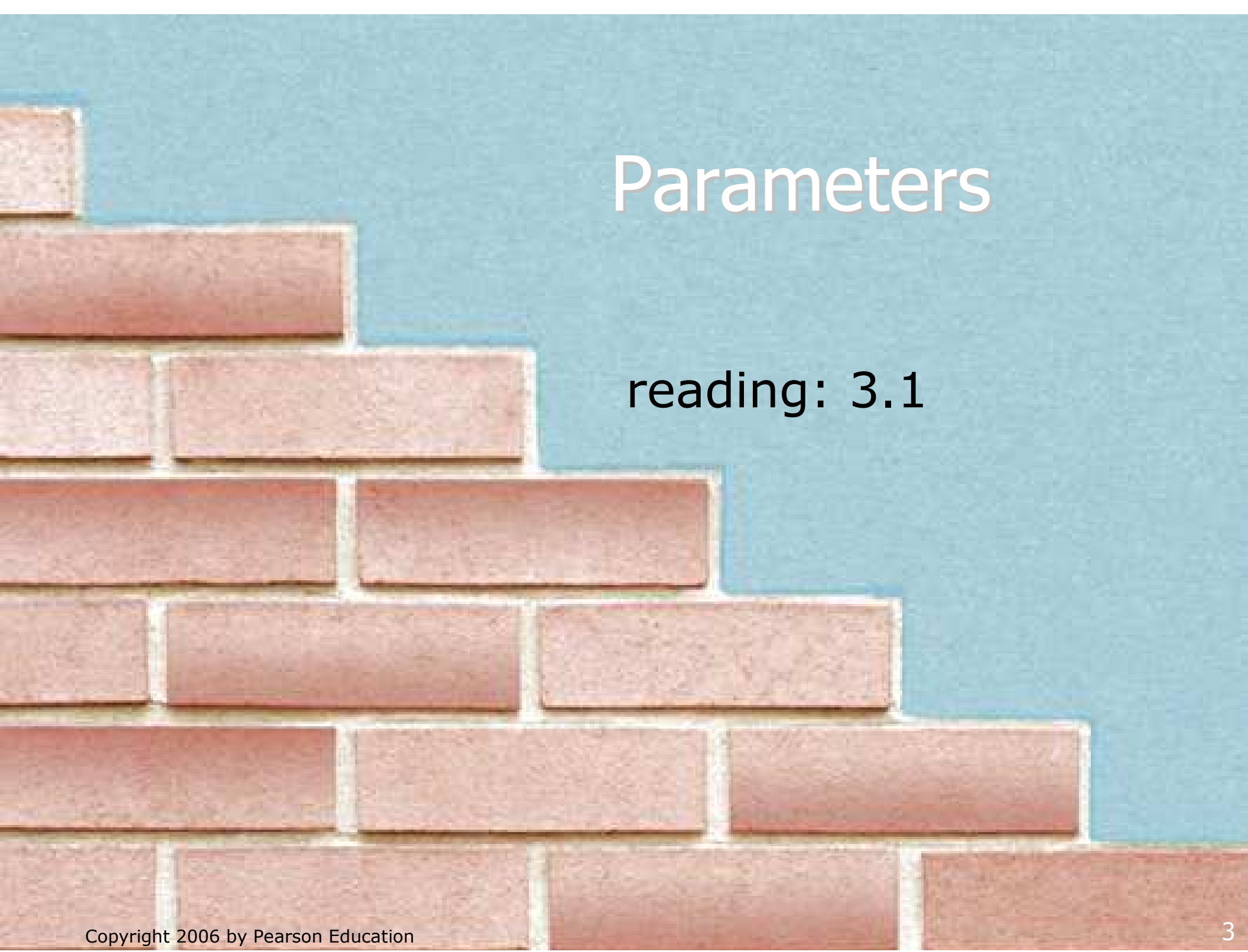


Lecture outline

- parameters
  - passing parameters to static methods
  - writing methods that accept parameters



Redundant recipes

- Recipe for baking **20** cookies:
  - Mix the following ingredients in a bowl:
    - **4** cups flour
    - **1** cup butter
    - **1** cup sugar
    - **2** eggs
    - **1** bag chocolate chips
    - …
  - Place on sheet and Bake for about 10 minutes.

- Recipe for baking **40** cookies:
  - Mix the following ingredients in a bowl:
    - **8** cups flour
    - **2** cups butter
    - **2** cups sugar
    - **4** eggs
    - **2** bags chocolate chips
    - …
  - Place on sheet and Bake for about 10 minutes.

Parameterized recipe

- Recipe for baking **40** cookies:
  - Mix the following ingredients in a bowl:
    - **8** cups flour
    - ...
- Recipe for baking **N** cookies:
  - Mix the following ingredients in a bowl:
    - **N/5** cups flour
    - **N/20** cups butter
    - **N/20** cups sugar
    - **N/10** eggs
    - **N/20** bags chocolate chips
    - ...
  - Place on sheet and Bake for about 10 minutes.

- **parameter**: A value that distinguishes similar tasks.
- **parameterize**: To express a set of similar tasks in a general way in terms of one or more parameters.

Redundant figures

- Consider the task of drawing the following figures:

```
**************

*******

**********************************

**********
*        *
**********

*****
*   *
*   *
*****
```

  - The lines and figures are similar, but not exactly the same.

A redundant solution

```
public class Stars1 {
    public static void main(String[] args) {
        drawLineOf13Stars();
        drawLineOf7Stars();
        drawLineOf35Stars();
        draw10x3Box();
        draw5x4Box();
    }

    public static void drawLineOf13Stars() {
        for (int i = 1; i <= 13; i++) {
            System.out.print("*");
        }
        System.out.println();
    }

    public static void drawLineOf7Stars() {
        for (int i = 1; i <= 7; i++) {
            System.out.print("*");
        }
        System.out.println();
    }

    public static void drawLineOf35Stars() {
        for (int i = 1; i <= 35; i++) {
            System.out.print("*");
        }
        System.out.println();
    }
    ...
```

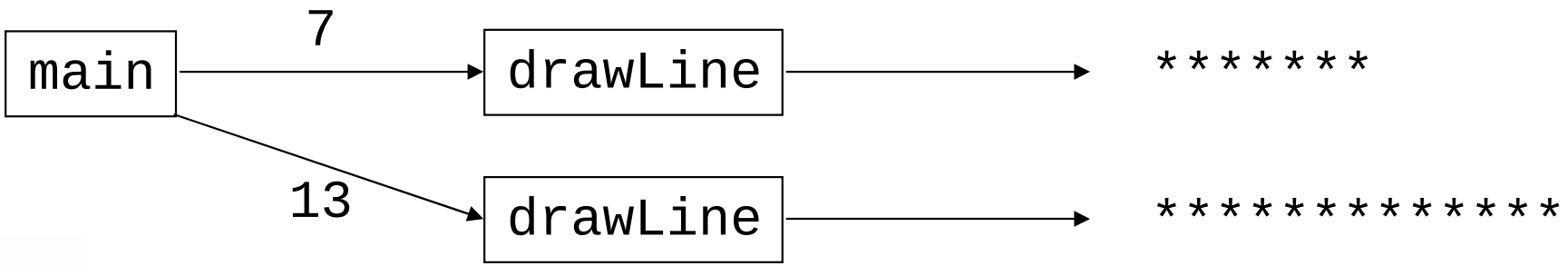
- The methods at left are redundant.
- Would constants help us solve this problem?
- What would be a better solution?
  - `drawLine` - A method to draw a line of any number of stars.
  - `drawBox` - A method to draw a box of any size.

Parameterization

- **parameterized method**: One that is given extra information (e.g. number of stars to draw) when it is called.

- **parameter**: A value passed to a method by its caller.

- Writing parameterized methods requires 2 steps:
  - *declare* the method to accept the parameter
  - *call* the method and pass the parameter value(s) desired



Declaring parameterized methods

- Parameterized method declaration syntax:

```
public static void <name> ( <type> <name> ) {
    <statement(s)> ;
}
```

- Example:

```
public static void printSpaces(int count) {
    for (int i = 1; i <= count; i++) {
        System.out.print(" ");
    }
}
```

  - Whenever `printSpaces` is called, the caller must specify how many spaces to print.

Calling parameterized methods

- **passing a parameter**: Calling a parameterized method and specifying a value for its parameter(s).

- Parameterized method call syntax:

```
<name> ( <expression> );
```

- Example:

```
System.out.print("*");
printSpaces(7);
System.out.print("**");
int x = 3 * 5;
printSpaces(x + 2);
System.out.println("***");
```

Output:

```
*       **                      ***
```

How parameters are passed

- When the parameterized method call executes:
  - the value written is copied into the parameter variable
  - the method's code executes using that value

```
public static void main(String[] args) {
    printSpaces(7);
    printSpaces(13);
}

public static void printSpaces(int count) {
    for (int i = 1; i <= count; i++) {
        System.out.print(" ");
    }
}
```

Value semantics

- **value semantics**: When primitive variables (`int`, `double`) are passed as parameters, their values are copied.
  - Modifying the parameter inside the method will not affect the variable passed in.

```
public static void main(String[] args) {
    int x = 23;
    strange(x);
    System.out.println("2. x = " + x);    // unchanged
    ...
}

public static void strange(int x) {
    x = x + 1;
    System.out.println("1. x = " + x);
}
```

Output:

```
1. x = 24
2. x = 23
```

Common errors

- If a method accepts a parameter, it is illegal to call it without passing any value for that parameter.

```
printSpaces();   // ERROR: parameter value required
```

- The value passed to a method must be of the correct type, matching the type of its parameter variable.

```
printSpaces(3.7);   // ERROR: must be of type int
```

- Exercise: Change the Stars program to use a parameterized static method for drawing lines of stars.

Stars solution

```
// Prints several lines of stars.
// Uses a parameterized method to remove redundancy.

public class Stars2 {
    public static void main(String[] args) {
        drawLine(13);
        drawLine(7);
        drawLine(35);
    }

    // Prints the given number of stars plus a line break.
    public static void drawLine(int count) {
        for (int i = 1; i <= count; i++) {
            System.out.print("*");
        }
        System.out.println();
    }
}
```

Multiple parameters

- Methods can accept multiple parameters.
  - When the method is called, you must pass values for each parameter.

- Multiple parameters declaration syntax:

```
public static void <name> ( <type> <name> ,
        <type> <name> , ..., <type> <name> ) {
    <statement(s)> ;
}
```

- Multiple parameters call syntax:

```
<name> ( <expression>, <expression>, ..., <expression> );
```

Multiple parameters example

```
public static void main(String[] args) {
    printNumber(4, 9);
    printNumber(17, 6);
    printNumber(8, 0);
    printNumber(0, 8);
}

public static void printNumber(int number, int count) {
    for (int i = 1; i <= count; i++) {
        System.out.print(number);
    }
    System.out.println();
}
```

Output:

```
444444444
171717171717

00000000
```

- Exercise: Write an improved Stars program that draws boxes of stars using parameterized static methods.

Stars solution

```
// Prints several lines and boxes made of stars.
// Third version with multiple parameterized methods.

public class Stars3 {
    public static void main(String[] args) {
        drawLine(13);
        drawLine(7);
        drawLine(35);
        System.out.println();
        drawBox(10, 3);
        drawBox(5, 4);
        drawBox(20, 7);
    }

    // Prints the given number of stars plus a line break.
    public static void drawLine(int count) {
        for (int i = 1; i <= count; i++) {
            System.out.print("*");
        }
        System.out.println();
    }
    ...
```

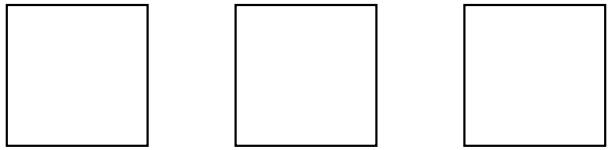
Stars solution, cont'd.

```
    ...

    // Prints a box of stars of the given size.
    public static void drawBox(int width, int height) {
        drawLine(width);

        for (int i = 1; i <= height - 2; i++) {
            System.out.print("*");
            printSpaces(width - 2);
            System.out.println("*");
        }

        drawLine(width);
    }

    // Prints the given number of spaces.
    public static void printSpaces(int count) {
        for (int i = 1; i <= count; i++) {
            System.out.print(" ");
        }
    }
}
```

Parameter "mystery" problem

- What is the output of the following program?

```
public class Mystery {
    public static void main(String[] args) {
        int x = 5, y = 9, z = 2;
        mystery(z, y, x);
        System.out.println(x + " " + y + " " + z);
        mystery(y, x, z);
        System.out.println(x + " " + y + " " + z);
    }

    public static void mystery(int x, int z, int y) {
        x++;
        y = x - z * 2;
        x = z + 1;
        System.out.println(x + " " + y + " " + z);
    }
}
```



Parameter questions

- Write a method named `printDiamond` that accepts a height as a parameter and prints a diamond figure:

```
  *
 ***
*****
 ***
  *
```

- Write a method named `multiplicationTable` that accepts a maximum integer as a parameter and prints a table of multiplication from `1 x 1` up to that integer times itself.
- Write a method named `bottlesOfBeer` that accepts an integer as a parameter and prints the "XX Bottles of Beer" song with that many verses.